Mini Project

RAM & ROM DESIGN USING VERILOG

By

VAIDEHI MULEY

M.Tech (VLSI Design)

VIT, Vellore

2022 - 24

INTRODUCTION

RAM (Random Access Memory)

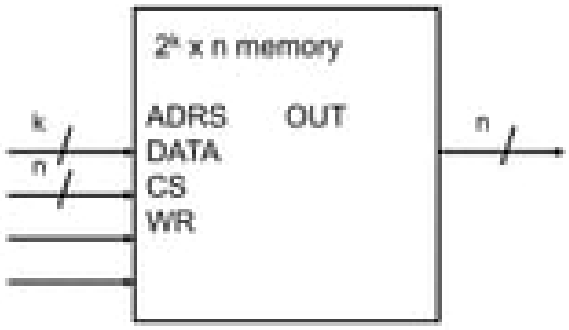

Fig.1.1 Block Diagram of RAM

Random Access Memory (RAM) is a fundamental component of modern computing systems, playing a crucial role in their performance and functionality. Unlike storage devices such as hard drives or SSDs, which store data persistently even when the power is off, RAM is volatile memory. This means it is used for temporary data storage while a computer is powered on, allowing the CPU to access data quickly and efficiently. Functionality: RAM serves as a high-speed temporary storage for data that the computer's processor (CPU) needs to access quickly. It holds data that is actively being used or manipulated by currently running programs importance lies in its role as a bridge between the CPU and long-term storage. It ensures that the CPU can quickly access the data it needs to execute instructions, significantly improving overall system performance. Insufficient RAM can lead to sluggish performance, frequent program crashes, or an inability to run certain applications altogether.

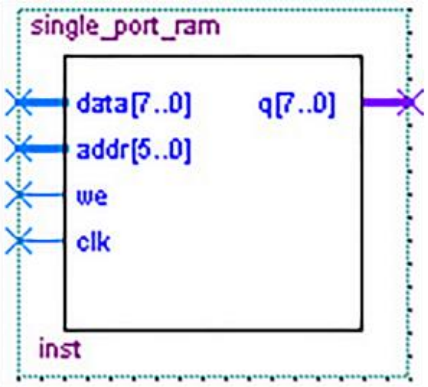

Fig.1.2 Block Diagram of Single port RAM

Single Port RAM (Random Access Memory) is a type of memory that allows data to be read from and written to a single port or interface at a time. It is a basic form of RAM commonly used in various applications where simple data storage and retrieval operations are sufficient.Structure: Single Port RAM typically consists of a single set of data input and output ports, which means it can handle either read or write operations at any given time, but not both simultaneously.Advantages: Single Port RAM is straightforward to design and implement, making it cost-effective for applications where basic data storage and retrieval operations are sufficient. It also consumes less power compared to more complex RAM types.Limitations: The primary limitation of single port RAM is its inability to perform simultaneous read and write operations. This can potentially lead to delays in data access if multiple devices or processes need to access the RAM at the same time.Many microcontrollers incorporate single port RAM for temporary data storage during program execution.Basic embedded systems used in appliances, industrial controls, or consumer electronics often utilize single port RAM for storing operational data.In applications where real-time data processing is not critical, single port RAM can handle data buffering and temporary storage effectively.

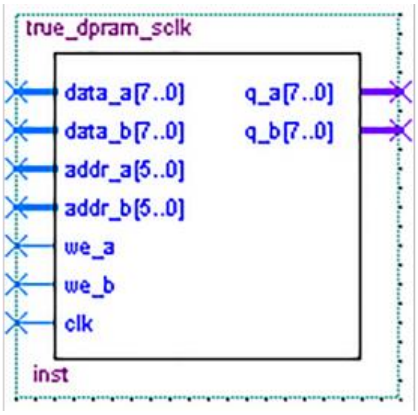

Fig.1.3 Block Diagram of Dual port RAM

Dual port RAM" (often abbreviated as DP RAM) is a type of memory that offers two separate ports or interfaces for simultaneous data access. This feature allows for independent read and write operations to occur simultaneously, which can be beneficial in certain applications requiring high-speed access from multiple sources.Dual Port Structur: Dual port RAM consists of two independent sets of data input and output ports, typically labeled as Port A and Port B. Each port operates independently, allowing for simultaneous read and write operations.Simultaneous Access: One of the primary advantages of dual port RAM is its ability to handle simultaneous data access from different devices or processes. For example, while one device is reading data from Port A, another device can be writing data to Port B concurrently. Advantages: ability to support multiple simultaneous access operations without conflict. This capability enhances system performance in applications where data throughput and real-time responsiveness are critical.Dual port RAM is used in routers, switches, and network interface cards (NICs) to handle simultaneous data transfers between network devices.In video processing systems, dual port RAM can manage simultaneous data read/write operations for video frames and image data, ensuring smooth playback and processing.Applications requiring real-time data processing, such as control systems in industrial automation or robotics, benefit from the simultaneous access capabilities of dual port RAM.

ROM (Read Only Memory)

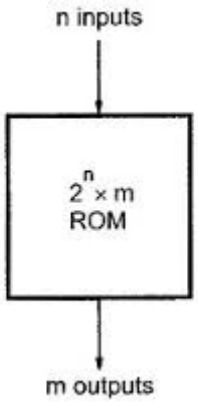

Fig.1.4 Block Diagram of ROM

ROM, or Read-Only Memory, is a type of computer memory that stores data and instructions that cannot be easily modified or overwritten by the user or the computer system. Unlike RAM (Random Access Memory), which is volatile and loses its data when power is turned off, ROM retains its contents even when the power is off. This characteristic makes ROM suitable for storing critical system firmware, boot loaders, and other essential programs that need to be preserved throughout the life of a device.Non-Volatile: ROM is non-volatile memory, meaning it retains its contents even when the power supply is turned off. This makes it ideal for storing permanent or semi-permanent data and instructions.Read-Only Access: As the name suggests, ROM typically allows data to be read from it, but not written to or modified by normal means. The contents are often programmed or "burned" into the memory during manufacturing and cannot be changed in the field.Applications: ROM is used for storing firmware, which includes the basic input/output system (BIOS) of computers, firmware of embedded systems (such as microcontrollers and IoT devices), and other critical software that needs to be permanently stored and easily accessible.ROM holds the BIOS or UEFI firmware, which initializes hardware components and loads the operating system during startup ROM is used in devices like smart TVs, smartphones, and IoT devices to store firmware and operating system kernels ROM is used in devices such as gaming consoles, digital cameras, and set-top boxes to store firmware and system software.

DESIGN & TESTBENCH USING VERILOG

RAM (Single Port)

//Design

```
module single_port_ram(
  input [7:0] data, //input data
  input [5:0] addr, //address
  input we, //write enable
  input clk, //clk
  output [7:0] q //output data
);

  reg [7:0] ram [63:0]; //8*64 bit ram
  reg [5:0] addr_reg; //address register

always @ (posedge clk)
    begin
      if(we)
        ram[addr] <= data;
      else
        addr_reg <= addr;
    end

  assign q = ram[addr_reg];
endmodule
```

//Testbench

```
module single_port_ram_tb;
  reg [7:0] data; //input data
  reg [5:0] addr; //address
  reg we; //write enable
  reg clk; //clk
  wire [7:0] q; //output data

  single_port_ram spr1(
    .data(data),
    .addr(addr),
    .we(we),
    .clk(clk),
    .q(q)
  );

  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1, single_port_ram_tb);

      clk=1'b1;
      forever #5 clk = ~clk;
```

```
  end

initial
  begin
    data = 8'h01;
    addr = 5'd0;
    we = 1'b1;
    #10;

          data = 8'h02;
    addr = 5'd1;
    #10;

    data = 8'h03;
    addr = 5'd2;
    #10;

    addr = 5'd0;
    we = 1'b0;
    #10;

    addr = 5'd1;
    #10;

    addr = 5'd2;
    #10;

    data = 8'h04;
    addr = 5'd1;
    we = 1'b1;
    #10;

    addr = 5'd1;
    we = 1'b0;
    #10;

    addr = 5'd3;
    #10;
  end

initial
  #90 $stop;

endmodule
```

RESULTS

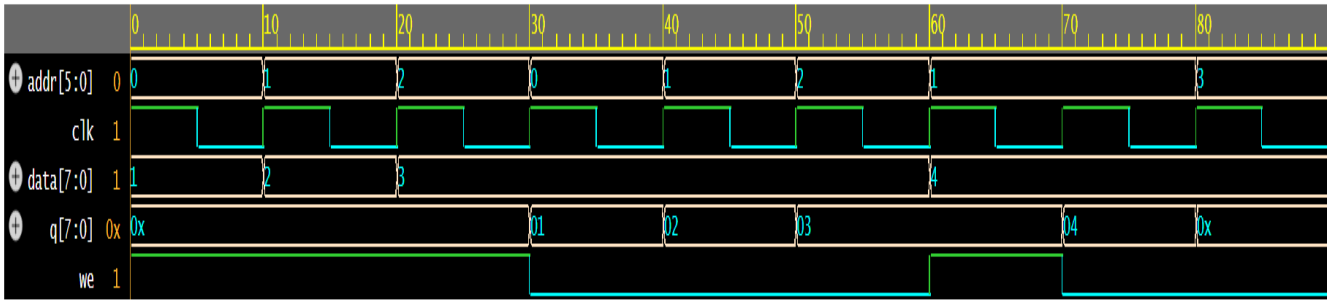

Fig.2.1 Waveform of Single port RAM

RAM (Dual Port)

//Design

```
module dual_port_ram(
  input [7:0] data_a, data_b, //input data
  input [5:0] addr_a, addr_b, //Port A and Port B address
  input we_a, we_b, //write enable for Port A and Port B
  input clk, //clk
  output reg [7:0] q_a, q_b //output data at Port A and Port B
);

  reg [7:0] ram [63:0]; //8*64 bit ram

 always @ (posedge clk)
   begin
     if(we_a)
       ram[addr_a] <= data_a;
     else
       q_a <= ram[addr_a];
   end

  always @ (posedge clk)
   begin
     if(we_b)
       ram[addr_b] <= data_b;
     else
       q_b <= ram[addr_b];
   end
  endmodul
```

// Testbench

```
module dual_port_ram_tb;
  reg [7:0] data_a, data_b; //input data
  reg [5:0] addr_a, addr_b; //Port A and Port B address
  reg we_a, we_b; //write enable for Port A and Port B
  reg clk; //clk
  wire [7:0] q_a, q_b; //output data at Port A and Port B

  dual_port_ram dpr1(
    .data_a(data_a),
    .data_b(data_b),
    .addr_a(addr_a),
    .addr_b(addr_b),
    .we_a(we_a),
    .we_b(we_b),
    .clk(clk),
    .q_a(q_a),
    .q_b(q_b)
  );

  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1, dual_port_ram_tb);
```

```
    clk=1'b1;
    forever #5 clk = ~clk;
  end

initial
  begin
    data_a = 8'h33;
    addr_a = 6'h01;

    data_b = 8'h44;
    addr_b = 6'h02;

    we_a = 1'b1;
    we_b = 1'b1;

    #10;

    data_a = 8'h55;
    addr_a = 6'h03;

    addr_b = 6'h01;

    we_b = 1'b0;
    #10;

    addr_a = 6'h02;

    addr_b = 6'h03;

    we_a = 1'b0;

    #10;

    addr_a = 6'h01;

    data_b = 8'h77;
    addr_b = 6'h02;

    we_b = 1'b1;
    #10;
  end
 initial
   #40 $stop;
endmodule
```

RESULTS

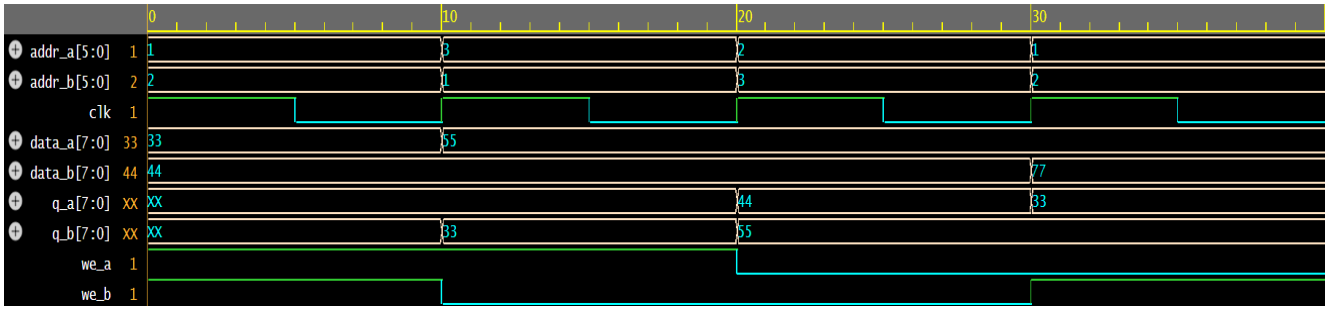

Fig.2.2 Waveform of Dual port RAM

ROM

```
//Design
module rom (
  input clk, //clk
  input en, //enable
  input [3:0] addr, //address
  output reg [3:0] data //output data
);

  reg [3:0] mem [15:0]; //4 bit data and 16 locations

  always @ (posedge clk)
    begin
      if (en)
        data <= mem[addr];
      else
        data <= 4'bxxxx;
    end

  initial
    begin
      mem[0] = 4'b0010;
      mem[1] = 4'b0010;
      mem[2] = 4'b1110;
      mem[3] = 4'b0010;
      mem[4] = 4'b0100;
      mem[5] = 4'b1010;
      mem[6] = 4'b1100;
      mem[7] = 4'b0000;
      mem[8] = 4'b1010;
      mem[9] = 4'b0010;
      mem[10] = 4'b1110;
      mem[11] = 4'b0010;
      mem[12] = 4'b0100;
      mem[13] = 4'b1010;
      mem[14] = 4'b1100;
      mem[15] = 4'b0000;
    end

endmodule
```

```
// ROM testbench

module rom_tb;
  reg clk; //clk
  reg en; //enable
  reg [3:0] addr; //address
  wire [3:0] data; //output data

  rom r1(
    .clk(clk),
    .en(en),
    .addr(addr),
    .data(data)
  );

  initial
    begin
```

```
    $dumpfile("dump.vcd");
    $dumpvars(1, rom_tb);

    clk=1'b1;
    forever #5 clk = ~clk;
  end

initial
  begin
    en = 1'b0;
    #10;

    en = 1'b1;
    addr = 4'b1010;
    #10;

    addr = 4'b0110;
    #10;

    addr = 4'b0011;
    #10;

    en = 1'b0;
    addr = 4'b1111;
    #10;

    en = 1'b1;
    addr = 4'b1000;
    #10;

    addr = 4'b0000;
    #10;
     addr = 4'bxxxx;
    #10;
  end

initial
  begin
    #80 $stop;
  end

endmodule
```

RESULTS

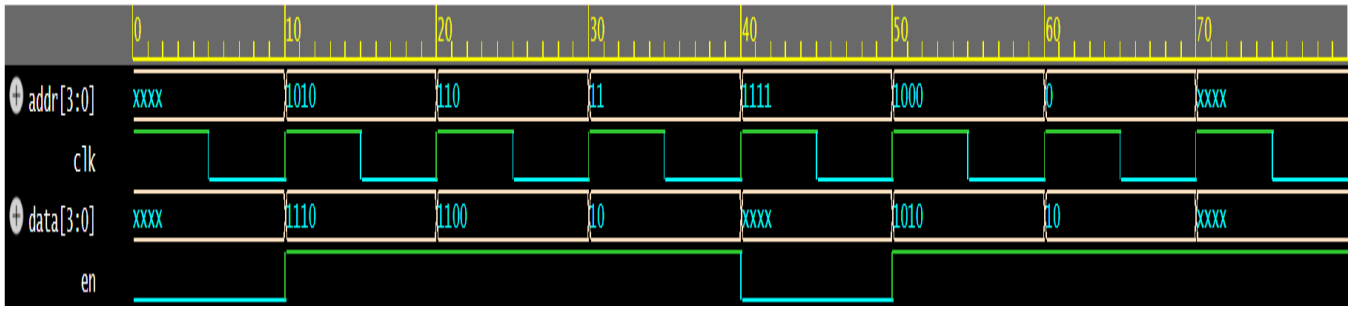

Fig.2.1 Waveform of ROM